ARMY RESEARCH LABORATORY


# Building Task-Oriented Applications: An Introduction to the Legion Programming Paradigm

**by Richard H Haney, Song J Park, and Dale R Shires**

**ARL-TR-7206** **February 2015**

## NOTICES

### Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

**Army Research Laboratory**
Aberdeen Proving Ground, MD 21005-5067

**ARL-TR-7206** **February 2015**

# Building Task-Oriented Applications: An Introduction to the Legion Programming Paradigm

**Richard H Haney**
**Oak Ridge Associated Universities**

**Song J Park and Dale R Shires**
**Computational and Information Sciences Directorate, ARL**

| REPORT DOCUMENTATION PAGE | Form Approved OMB No. 0704-0188 |
|---|---|



| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED (From - To) |
|---|---|---|
| February 2015 | Final | July 1, 2014–October 28, 2014 |

**4. TITLE AND SUBTITLE**
Building Task-Oriented Applications: An Introduction to the Legion Programming Paradigm

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**
Richard H Haney, Song J Park, and Dale R Shires

**5d. PROJECT NUMBER**

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
US Army Research Laboratory
ATTN: RDRL-CIH-S
Aberdeen Proving Ground, MD 21005-5067

**8. PERFORMING ORGANIZATION REPORT NUMBER**
ARL-TR-7206

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

**10. SPONSOR/MONITOR'S ACRONYM(S)**

**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
Approved for public release; distribution is unlimited.

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**
The Legion programming system from Stanford University was developed to address the specific needs of portable parallel heterogeneous programming with emphasis on program data. Data-centric Legion abstracts the underlying hardware such that the focus is algorithmic development rather than the idiosyncrasies of different computing architectures. Legion is part of a small but growing movement to treat parallel programming design and development in a task-oriented fashion and, as of this writing, is the only one to address this paradigm dynamically. This work employs a gravitational n-body simulation as a vehicle to introduce Legion to a wider audience of parallel programmers.

**15. SUBJECT TERMS**
n-body, software design, Legion, task-oriented, parallel programming

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | Richard H Haney |
| Unclassified | Unclassified | Unclassified | UU | 18 | 19b. TELEPHONE NUMBER (*Include area code*) 410-278-7866 |

**Standard Form 298 (Rev. 8/98)**
Prescribed by ANSI Std. Z39.18

## Contents

## List of Figures



## List of Tables

## 1. Introduction

Efforts to leverage the growing computational power available using parallel programming inexorably must address the significant factor of heterogeneous computing.[1–3] Multicore central processing units (CPUs) and various hardware accelerators exacerbate a complex architectural landscape that inevitably constrains parallel application design and implementation.[4] With the underlying hardware changes, code must often be repurposed to remain viable. Achieving high performance on heterogeneous systems with complex memory hierarchies proves to be a major challenge. Furthermore, as data movement dictates computational cost and power, efficiency on future systems will require programming models with program data reasoning.

The Legion programming system from Stanford University was developed to address the specific issues involved with parallel heterogeneous computing from a data-centric viewpoint.[5] Legion defines a methodology whereby the developer can view parallelism through a grouping of specially designated functions called tasks, each of which will operate on some portion of the global domain in parallel.[6] This parallelism invoked by Legion is often referred to as task-parallelism and is implicit in the design.[5,6]

However, to take advantage of the benefits afforded by Legion for parallel heterogeneous computing it is necessary to understand how the programming system works. The intent of this study is to examine the syntax and application building potential of the Legion programming system via an emblematic computing algorithm. The n-body problem, an important part of many physics applications,[7–9] was chosen as the vehicle to examine Legion in this regard. An outline of the rest of this work follows.

Section 2 provides the reader with a background into both the n-body simulation utilized by this work and the Legion programming system. This section also describes the technical specifications of the computing environment used by this study. The details of building the n-body simulation using Legion are discussed in Section 3. Section 4 discusses the observed results, and the final section provides a conclusion and potential future directions.

## 2. Background

This section will briefly provide a background into the n-body problem as well as some pertinent information regarding the Legion programming system. Legion is a fairly flexible system that allows developers to create initial programs in a rather facile manner but has the ability to evolve into far more complex structures and algorithms. Therefore, the information regarding Legion

in this work is far from exhaustive, and the interested reader is referred to Bauer et al[5,6] and Treichler[10] for more in-depth examinations of particulars beyond the implementation of the n-body algorithm used in this work.

### 2.1 The N-body Problem

The n-body problem describes an approach to understand how a set of n-bodies/particles interact with one another over time and is based on principles proposed by Sir Isaac Newton.[11] The n-body problem is a critical component to many applications but generally falls under 2 paradigms for computational modeling: pair-wise or hierarchical tree-based solutions.[12–14] This work follows the pair-wise computational model for gravitational simulation with an asymptotic computational complexity of $O(N^2)$ where $N$ is the number of bodies/particles in the system[15] as performance is not the intent. As illustrated in Fig. 1, the pair-wise algorithm generates this computational cost via the body-to-body interactions calculated in lines 6–17.

```
1.    SET gdt TO GRAVITY * TimeStep
2.    SET EPS TO 0.0001
3.    FOR T <= 0 TO TotalTime
4.      FOR I <= 0 TO N
5.          Particle p0 <= Particles[I]
6.          FOR J <= I + 1 TO N
7.               Particle p1 <= Particles[J]
8.               dx <= p1.x – p0.x
9.               dy <= p1.y – p0.y
10.              dz <= p1.z – p0.z
11.              invr <= 1 / SQRT(dx*dx + dy*dy + dz*dz + EPS)
12.              invr3 <= invr*invr*invr
13.              force <= p1.mass*invr3
14.              ax <= ax + force*dx
15.              ay <= ay + force*dy
16.              az <= az + force*dz
17.         END FOR
18.         Xnew[I] <= p0.x + gdt*p0.velX + 0.5*gdt*gdt*ax
19.         Ynew[I] <= p0.y + gdt*p0.velY + 0.5*gdt*gdt*ay
20.         Znew[I] <= p0.z + gdt*p0.velZ + 0.5*gdt*gdt*az
21.         p0.velX <= p0.velX + gdt*ax
22.         p0.velY <= p0.velY + gdt*ay
23.         p0.velZ <= p0.velZ + gdt*az
24.     END FOR
25.     FOR I <= 0 TO N
26.         Particle p <= Particles[I]
27.         p.x <= Xnew[I]
28.         p.y <= Ynew[I]
29.         p.z <= Znew[I]
30.     END FOR
31.   END FOR
```

Fig. 1 Pair-wise n-body algorithm

## 2.2 The Legion Programming Model

The Legion programming system addresses parallelism in a data-centric fashion, providing a set of tasks to execute on defined logical regions in the code.[5,6,10] These tasks are structured in a hierarchy and will bind associated logical regions to actual physical addresses at runtime to allow for optimal system flexibility. A top-level task is defined for every Legion program but beyond this the developer is free to define any number of custom operations as long as they are registered by the system prior to execution.[5,10]

## 2.3 Registering Legion Tasks

The top-level task is executed by the Legion programming system first and can be viewed as the control point for the system. Figure 2 shows the general hierarchy of Legion task registration whereby the top-level task is first followed by any other tasks for the application before the invocation of the Legion start function.[5,6]

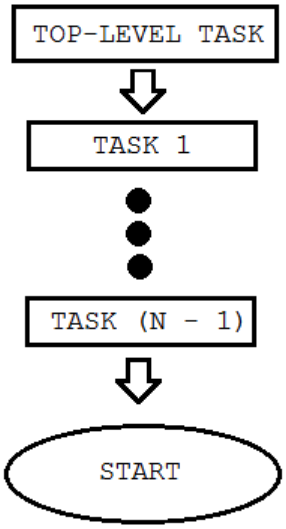


Fig. 2 Legion task registration hierarchy

The registration of Legion tasks with the runtime engine is accomplished by passing the defined task to the template of the register-Legion-task object along with a unique integer identifier.[5,6] This allows the Legion runtime to associate the defined task with the globally unique identification value.[6] Once the defined tasks are properly registered with the Legion runtime, the system must start execution.

The Legion start function will automatically call all underlying communication libraries such as the Message Passing Interface (MPI) to properly initialize the system.[5,6,10] The start function will look for all defined and registered Legion tasks and execute them in the predefined hierarchy (see Fig. 2). Upon completion, Legion will automatically call underlying communication libraries clean-up/shut-down operations returning the resulting flag (e.g., success or failure).[6]

## 2.4 Launching Legion Tasks

The defined and registered Legion tasks, with the exception of the top-level task, must be explicitly invoked via the TaskLauncher object.[5] TaskLauncher objects can be passed either single or multiple arguments with the TaskArgument or ArgumentMap objects, respectively.[6] However, to ensure optimal flexibility and performance, Legion follows the delayed execution model and does not bind arguments until the TaskLauncher object is passed to the actual execution point in the program.[5,10]

## 2.5 Domain of Execution

The domain of execution for a Legion program is defined as the Domain object such that structured and unstructured versions are available.[5,6,10] The unstructured domain is a grouping of points in the logical region that the developer must explicitly call and initialize, whereas the structured domain is represented by a rectangular object that is implicitly initialized.[6] These domain definitions are validated prior to execution and represent logical regions that each task can access and manipulate as per the dictates of the task privileges and coherence settings.[5,6,10]

The logical regions in Legion are broken down into sets of index spaces and field spaces.[5] The index space can be views as a row in row-column descriptions and the field space as columns such that the field space is an applied description of entries in the current region.[6] These regions can be shared by other regions or can maintain exclusivity as shown in Fig. 3.

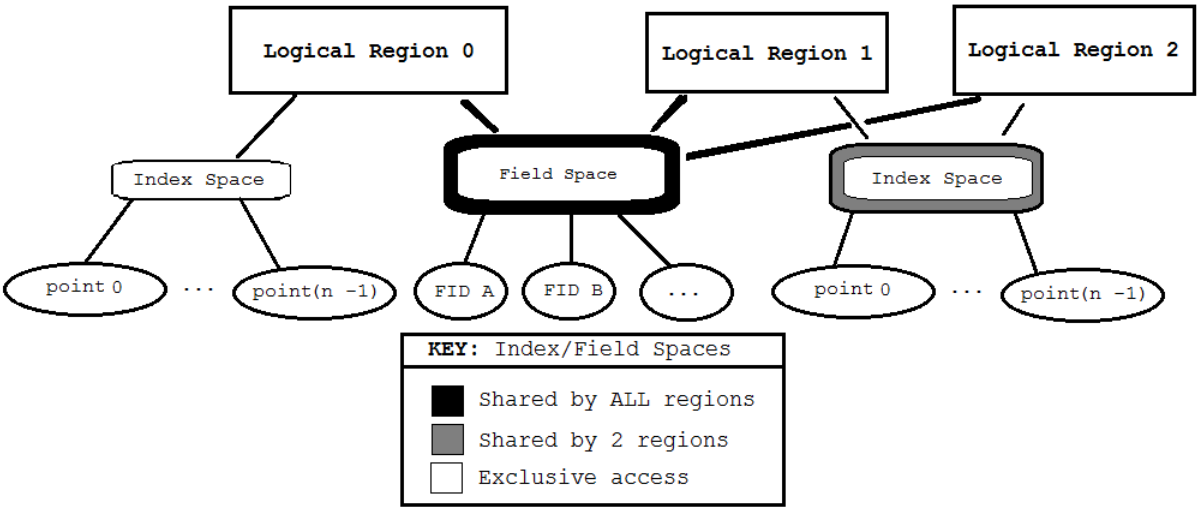


Fig. 3 Legion region descriptions

## 2.6 Legion Is Object-Oriented

Legion supports covariant and contra-covariant bindings, meaning that the order of sub-types (e.g., polymorphism) is preserved or could be reversed, respectively. The ability to function in this manner ensures that Legion will execute in an object-oriented manner that any C++ programmer should be familiar with. These arguments are passed in and out of Legion tasks in a pass-by-value manner.

Legion permits a given task to have declared side effects on declared regions of data, resulting in the ability of logical regions to act as heaps with tasks themselves changing state. The design of Legion allows task to manifest as imperative or functional provided maximum flexibility within the context of a C/C++ program. However, this design means that Legion views function pointers as analogous to global variables and therefore the passing function pointers around to tasks is strongly discouraged.

## 2.7 Computing Environment Employed

The computing environment used for this work is a 64-node heterogeneous cluster consisting of 48 IBM dx360M4 nodes, each with one Intel Phi 5110P and 16 dx360M4 nodes each with one NVIDIA Kepler K20M graphics processing unit. Each node contained dual Intel Xeon E5-2670 (Sandy Bridge) CPUs, 64 GB of memory and a Mellanox FDR-10 Infiniband host channel adaptor.[16,17]

However, this work does not employ accelerators, concentrating instead on the behavior of a single compute node, e.g., Intel Xeon E5-2670 limited to a single processing core. The application of the Legion programming model and runtime library to a single CPU job slot isolates the shared memory behavior, which could then be extended to the distributed parallel environment at a later date.

Workload management for the computing environment was implemented using IBM Platform LSF 9.1, which is specifically designed for mission-critical high-performance computing environments.[18] The operation was limited to a single, nonaccelerator, job slot by setting the batch submit switch $n$ to 1 and not explicitly calling any Intel Many Integrated Core Architecture devices.[18] The BLAUNCH command was employed to allow parallel operation within the shared memory paradigm for this work.[18]

The next section discusses the implementation of the gravitational n-body simulation using the Legion programming system.

## 3. Building the Application

The first step in building the n-body simulation employed by this study is to properly install the Legion programming system itself as described by Harada.[2] Legion can be deployed as either a shared or distributed memory system; the latter requires using the GASNet library for defined communication calls, and the reader is referred to Harada[2] for further details. After downloading and setting the Legion runtime environment variable(s), as per Harada,[2] the next step is to decide how to partition the domain as a set of Legion tasks.

### 3.1 Defining Tasks

Legion operates the set of tasks as a hierarchy with the top-level task the designated control point[5]; the n-body simulation actualized for this study is no exception. The top-level task for this work employs an unstructured set of bodies/particles and must therefore be explicitly initialized before validation. This top-level task will instantiate a logical region that will then be loaded with data from an external file as a structure of particles using a defined inline TaskLauncher object (see Fig. 4). Once complete, the physical region is unmapped so that control may then pass to the next task in the hierarchy, i.e., the main task.

```
1.    SET Particles p TO NULL         // array of particle structs
2.    CALL load_file_data (file)      // function loads external file
3.    CREATE Logical Region lr        // bind logical region
4.    SET ptr_t next TO 0             // base of particles
5.    CREATE TaskLauncher (WRITE)     // task launcher
6.    FOR I <= 0 TO N
7.         SET ptr TO next->value++   // next object in particle set
8.         WRITE ptr (p [I])
9.    END FOR
10.   UNMAP REGION
11.   CALL main-task
```

Fig. 4 Top-level task

The main task defined for this work controls the outer loop of the n-body algorithm shown as line 3 in Fig. 1. Essentially the only operation this task performs is to call the last task in the application hierarchy, i.e., the force-update task (see Fig. 5). The main task must unmap and then remap the current physical region when calling the force-update task as this will allow Legion to properly schedule execution independent of the enclosing task.

**Listing 3.**

```
1.    FOR T <= 0 TO TOTAL TIME
2.          UNMAP REGION
3.          CREATE TaskLauncher (force-update, READ-WRITE)
4.          CALL force-update     // sub-task
5.          REMAP REGION
6.    END FOR
```

Fig. 5 Main task

The force-update task for this work is derived directly from lines 6–17 and 18–23 of Fig. 1 and, as such, is exactly the same algorithm. Therefore, for the sake of brevity, it is not repeated at this point.

### 3.2 Final Algorithm

Once the task partitioning strategy has been designed, the final program is ready to deploy and can be seen in Fig. 6.

```
1.    CALL top-level-task
2.          LOAD input data
3.          WRITE particle data to PHYSICAL_REGION
4.    CALL main-task
5.          FOR EACH Time Step
6.               CALL force-update-task
7.          END FOR
8.    DONE
```

Fig. 6 Final application design

## 4. Observed Results

The objective of this work is a better understanding of the practical uses of the Legion programming system with regards to an example computing application rather than performance. As such, it is critical that the validity of the program itself be established. This section discusses the actual observed results of the gravitational n-body application gathered using Legion against those from a standard C++ implementation. Both of these programs are executed against the same input files and computing environments.

The data in the Table shows the coordinate position of the C++ version of the gravitational n-body program used in this study with that of the corresponding Legion application. These coordinates are for an 8-body system and clearly show the correctness of the program.

Table N-body validity for 8 bodies

| Particle | X | Y | Z | X-legion | Y-legion | Z-legion |
|---|---|---|---|---|---|---|
| 0 | 1.66 | 9.37 | 6.211 | 1.66 | 9.37 | 6.211 |
| 1 | 9.19 | 3.94 | 7.59 | 9.19 | 3.94 | 7.59 |
| 2 | 9.36 | 0.46 | 0.37 | 9.36 | 0.46 | 0.37 |
| 3 | 6.70 | 8.34 | 5.66 | 6.70 | 8.34 | 5.66 |
| 4 | 5.26 | 9.94 | 9.12 | 5.26 | 9.94 | 9.12 |
| 5 | 10.00 | 0.00 | 10.00 | 10.00 | 0.00 | 10.00 |
| 6 | 8.60 | 8.39 | 8.77 | 8.60 | 8.39 | 8.77 |
| 7 | 0.00 | 10.00 | 0.00 | 0.00 | 10.00 | 0.00 |

## 5. Conclusions and Future Work

This work has presented the Legion programming paradigm within the context of an important computational algorithm employed in many of today's high performance computing research, the n-body simulation. The gravitational n-body algorithm was defined using the Legion programming paradigm to illustrate the potential for future research in computing and has been shown to be both practical and correct for this purpose. The Legion programming system is flexible and introduces the developer to task-oriented programming, free from underlying computing architectures and communication procedures.

Future directions for this programming model are numerous and include a more effective and portable design for parallel heterogeneous computing. In particular the extension of this work to include the Fast Multipole Method (FMM) of the n-body problem is an interesting direction. FMM is currently being planned in concert with Stanford University as a part of a larger strategy of task-oriented solutions to heterogeneous parallel computing.

## 6. References


1. Topcuoglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems. 2002;13(3):260–274.

2. Harada T. Heterogeneous particle-based simulation. SIGGRAPH Asia 2011 Sketches. Proceedings of The International Conference on Computer Graphics and Interactive Techniques; 2011 Aug 7–11; Vancouver, Canada. New York (NY): ACM; c2011. Article 19.

3. Lashuk I, Chandramowlishwaran A, Langston H, Nguyen TA, Sampath R, Shringarpure A, Vuduc R, Ying L, Zorin D, Biros G. A massively parallel adaptive fast-multipole method on heterogeneous architectures. In: Proceedings of the 2009 ACM/IEEE International Conference for High Performance Computing Networking, Storage and Analysis; 2009 Nov 14–19; Portland, OR; New York (NY): ACM; c2009. Article No. 58.

4. Phothilimthana P, Ansel MJ, Ragan-Kelley J, Amarasinghe S. Portable performance on heterogeneous architectures. In: Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems; 2013 Mar 16–20; Houston, TX. New York (NY): ACM; c2013. p. 431–444.

5. Bauer M, Treichler S, Slaughter E, Aiken A. Legion: expressing locality and independence with logical regions. Proceedings of SC '12 - The International Conference for High Performance Computing, Networking, Storage and Analysis; 2012 Nov 10–16; Salt Lake City, UT. Los Alamitos (CA): IEEE Computer Society Press; c2012. Article No. 66.

6. Bauer M, Treichler S, Slaughter E, Aiken A, McCormick P. Legion programming system. [accessed 2014 Jun 15]. http://legion.stanford.edu/. GitHub 2014 Jan 6. [Online].

7. Fitch BG, Rayshubskiy A, Eleftheriou M, Ward T, Giampapa M, Pitman MC, Germain R. Blue matter: approaching the limits of concurrency for classical molecular dynamics. Proceedings of the 2006 ACM/IEEE, Conference on Supercomputing; 2006 Nov 11–17; Tampa, FL. New York (NY): ACM; c2006. Article No. 87.

8. O'Shea BW, Bryan G, Bordner J, Norman ML, Abel T, Harkness R, Kritsuk A. Introducing Enzo, an AMR cosmology application, in adaptive mesh refinement - theory and applications. Chicago (IL): Springer Berlin Heidelberg; c2005. p. 341–349. Vol. 41.

9. Salmon J. Parallel N log N N-body algorithms and applications to astrophysics. In: Compcon Spring '91. Digest of Papers; 1991, San Francisco, CA.

10. Treichler S, Bauer M, Aiken A. Language support for dynamic, hierarchical data partitioning. In: Proceedings of the 2013 ACM SIGPLAN international conference on object oriented programming systems languages and applications; 2013 Indianapolis, IN. New York (NY): ACM; c2013. p. 495–514

11. Trenti M, Hut P. N-body simulations. Scholarpedia. 2008;3(5):3930.

12. Board J, Schulten K. The fast multipole algorithm. Computing in Science and Engineering. 2000;76–79.

13. Nyland L, Harris M, Prins J. Fast N-Body Simulation with CUDA. GPU GEMS. 2007;3(1): 677–696.

14. Grama AY, Kumar V, Sameh A. Scalable parallel formulations of the barnes-hut method for n-body simulations. In: Supercomputing '94. Proceedings of the 1994 ACM/IEEE Conference on Supercomputing; 1994 Nov 14–18; Washington, DC. Los Alamitos (CA): IEEE Computers Society Press; c1994. p. 439–448.

15. Reif JH, Tate SR. The complexity of N-body simulation. In: Automata, languages and programming. Lund (Sweden): Springer Berlin Heidelberg; 2005. p. 162–176.

16. Intel Corporation, Intel Xeon Processor E5-1600 / E5-2600/E5-4600 Product Families Datasheet - Volume 1 and 2, Intel, 2012.

17. Intel Corporation. Intel Xeon Phi Coprocessor Instruction Set Architecture Reference Manual, 7 Sep 2012. [accessed 2014 Jun 1]. https://software.intel.com/en-us/mic-developer.

18. IBM Corporation. IBM Platform LSF V9.1.2 documentation, [accessed 2014 Jun]. 1992–2013 Jan 1 [Online]. Available: http://www-01.ibm.com/support/knowledgecenter /#!/SSETD4_9.1.2/lsf_welcome.html.

| | |
|---|---|
| 1 (PDF) | DEFENSE TECHNICAL INFORMATION CTR DTIC OCA |
| 2 (PDF) | DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT |
| 1 (PDF) | GOVT PRINTG OFC A MALHOTRA |
| 1 (PDF) | DIR USARL RDRL CIH S R HANEY |

INTENTIONALLY LEFT BLANK.